Библиотеки

Manim

видос 1 blue 3 brown про то как он визуализирует его вдиосы.

<https://www.youtube.com/watch?v=rbu7Zu5X1zI&t=12s>

A behind-the-scenes look at how I animate videos.

Code for all the videos: [https://github.com/3b1b/videos](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbC1BU3hMQmQwZk9MbWZ3LUlRZ2hXOWZ2Ul9sUXxBQ3Jtc0trTFhQLWFxOHlGaVBpTDN5dFJBcTk1Y3FHaHBxVmNpbmp3bVpSUnd0akdlRk1FOVJOcFpyY2ZyNlRKa2haVTdFVUpFa002SmdPS3h2Z3dRUlRudGRPbjA5Z2IxQTd0cnlTLXJqV2hMbWdubHdidGI2aw&q=https%3A%2F%2Fgithub.com%2F3b1b%2Fvideos&v=rbu7Zu5X1zI)

Manim: [https://github.com/3b1b/manim](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbHpMX1Z2eWdzVUQ4ZjVwanM4dURBbmpQYVk2UXxBQ3Jtc0tubGNXX3pVRDRjMUVuQkk2TWt3Z3JLNDV6QXgxZ3hQR2I2SUtNSXNZNXBCSUJDN2ZmUXVQaGlITUVVU0ZSMDVpVmtWNzVteV9lZGtnSWg4aXRaVi16WFV6VkwwMmFqbkRGNWI2WGdIdmZHSEJaSC1uSQ&q=https%3A%2F%2Fgithub.com%2F3b1b%2Fmanim&v=rbu7Zu5X1zI)

Community edition: [https://github.com/ManimCommunity/manim/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbVAySmtLenN4RFM2Z2JMRXViX0VJLUprMGtpUXxBQ3Jtc0trdVkwZk5DUVByRzRNckhIZHFLeXR5VEZfckJWY1Y0LWxlN2gxcVpRZU81RS1xSFFyWDNHTmx5SXJBeHpobHRlTWpfS3JvQ2cxWC13aXdrQzdqYjVsUmV1WTRZZWFrOFpiQUtzQXRRNDNYaUR6Vzl2UQ&q=https%3A%2F%2Fgithub.com%2FManimCommunity%2Fmanim%2F&v=rbu7Zu5X1zI)

Example scenes shown near the end: [https://github.com/3b1b/manim/blob/ma...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbG95ak5YQ2w4cjJRLWN5UGE3SjQwRVNHQ3lId3xBQ3Jtc0tuRndMSHlUUUVpNG5VbGNZWXlEbnlINDEtQUs0bFVkdnozOXpJSXZGWHFFVE9xc3hkeGg2OFhYVUdnNHdTTEJMbzhad016MzJYYzlYWEJQMDFrWVlfMmtPYk5nNW1UT2ZGRmMtTzBRanBhSldGQzFhdw&q=https%3A%2F%2Fgithub.com%2F3b1b%2Fmanim%2Fblob%2Fmaster%2Fexample_scenes.py&v=rbu7Zu5X1zI)

I added some more details about the workflow shown in this video to the readme of the videos repo: [https://github.com/3b1b/videos?tab=re...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqa0xfdExMSlBoenFLYVlIV3ZQTFhtNEs5MG1LZ3xBQ3Jtc0tremp5SVhtTnJXR0R3YkxJX2dxcmNqdnNrbWpIMG1KLVFlVlp1enJ0S3ZNREtQQm94cHdoaTNSQXRaQlpEZkpobGh1Y0ZFZnJzVk1XQmtoY3k0aXJpVlNvcVA4ZXZRWVpXRjN3R0JIZHN3VkI1VDZ2SQ&q=https%3A%2F%2Fgithub.com%2F3b1b%2Fvideos%3Ftab%3Dreadme-ov-file%23workflow&v=rbu7Zu5X1zI)

These lessons are funded directly by viewers: [https://3b1b.co/support](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbHZURVh6UWQ4eV9fZ1pRZEZnLWZoUnFoR0RUUXxBQ3Jtc0tseWpSOHBIS0J6YThMYjVuUERYclFBVXhYYWJHU0ZtZV83ZWFrM2R6ekMzSGRkT3BJYnFkLUdyOUlGRVd5Z25oRjZCVVVpV3Z1RzVYTHZtSVV2M1hDMTgzNDZqMkRFcHlZV28wUGtDZDVZbjkxQTJldw&q=https%3A%2F%2F3b1b.co%2Fsupport&v=rbu7Zu5X1zI)

2д графики и интервалы

# Плотность распределения вероятности

import numpy as np

import matplotlib.pyplot as plt

np.random.seed(7)

size = 1000

data = np.random.normal(loc=6.5, scale=2, size=size)

plt.figure(figsize=(10, 2))

plt.scatter(data, np.zeros\_like(data), alpha=0.6, color='blue', s=10)

plt.xlabel("Х(часов)", fontsize=12)

plt.yticks([])

plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.show()

#Определяем интервалы

bin\_edges = np.linspace(min(data), max(data), num=10) # 10 равных интервалов

counts, \_ = np.histogram(data, bins=bin\_edges) # Подсчёт количества значений в интервалах

# Центры интервалов для отображения

bin\_centers = (bin\_edges[:-1] + bin\_edges[1:]) / 2

# Построение графика

plt.figure(figsize=(10, 4))

plt.scatter(data, np.zeros\_like(data), alpha=0.6, color='blue', s=10, label='Данные') # Облако точек

plt.stem(bin\_centers, counts, linefmt='r-', markerfmt='ro', basefmt=" ", label='Количество в интервалах') # Частоты

# Оформление

plt.xlabel("Х (часы)", fontsize=12)

plt.ylabel("Количество", fontsize=12)

plt.title("Распределение данных по интервалам", fontsize=14)

plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.legend()

plt.tight\_layout()

# Показ графика

plt.show()

interval\_lengths = np.diff(bin\_edges)

density = counts / ( interval\_lengths \* size )

print(sum(density))

# Задаём интервалы с разной длиной

bin\_edges = [2, 5, 6, 7, 9, 12] # Интервалы разной длины

counts, \_ = np.histogram(data, bins=bin\_edges) # Подсчёт количества значений в интервалах

# Центры интервалов для отображения

bin\_centers = [(bin\_edges[i] + bin\_edges[i + 1]) / 2 for i in range(len(bin\_edges) - 1)]

# Построение графика

plt.figure(figsize=(10, 4))

plt.scatter(data, np.zeros\_like(data), alpha=0.6, color='blue', s=10, label='Данные') # Облако точек

plt.stem(bin\_centers, counts, linefmt='r-', markerfmt='ro', basefmt=" ", label='Количество в интервалах') # Частоты

# Оформление

plt.xlabel("Х (часы)", fontsize=12)

plt.ylabel("Количество", fontsize=12)

plt.title("Распределение данных по интервалам разной длины", fontsize=14)

plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.xticks(bin\_edges) # Метки интервалов

plt.legend()

plt.tight\_layout()

# Показ графика

plt.show()

# Задаём интервалы с разной длиной, но с нормировкой и плотностью

bin\_edges = [2, 5, 6, 7, 9, 12] # Интервалы разной длины

counts, \_ = np.histogram(data, bins=bin\_edges) # Подсчёт количества значений в интервалах

#нормализация и плотность

interval\_lengths = np.diff(bin\_edges)

counts = counts / ( interval\_lengths \* size )

# Центры интервалов для отображения

bin\_centers = [(bin\_edges[i] + bin\_edges[i + 1]) / 2 for i in range(len(bin\_edges) - 1)]

# Построение графика

plt.figure(figsize=(10, 4))

plt.scatter(data, np.zeros\_like(data), alpha=0.6, color='blue', s=10, label='Данные') # Облако точек

plt.stem(bin\_centers, counts, linefmt='r-', markerfmt='ro', basefmt=" ", label='Количество в интервалах') # Частоты

# Оформление

plt.xlabel("Х (часы)", fontsize=12)

plt.ylabel("Количество", fontsize=12)

plt.title("Распределение данных по интервалам разной длины", fontsize=14)

plt.grid(axis='x', linestyle='--', alpha=0.7)

plt.xticks(bin\_edges) # Метки интервалов

plt.legend()

plt.tight\_layout()

# Показ графика

plt.show()

print(sum(counts))

Нейросети

# Анимация выхода нейрона от весов

class FCNN(torch.nn.Module):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.fc1 = torch.nn.Linear(1,1)

self.act = torch.nn.ReLU()

self.act = torch.nn.LeakyReLU()

self.act = torch.nn.Sigmoid()

#self.act = torch.nn.Tanh()

def forward(self,x):

x = self.fc1(x)

x = self.act(x)

return x

model = FCNN()

print(model)

print(model(torch.randn(10,1)))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

# Генерируем данные

x = torch.linspace(-100, 100, 1000).reshape(-1, 1)

state\_dict = model.state\_dict()

# Подготовка графика

fig, ax = plt.subplots(figsize=(6, 6))

ax.set\_xlim(-100, 100)

ax.set\_ylim(-2, 2)

ax.set\_xlabel("x")

ax.set\_ylabel("y")

sc = ax.scatter([], []) # Пустой scatter plot

# Функция для обновления данных

def update(frame, w1,b1, text\_w\_b):

a = 0

b = a + 16

if a <= frame < b:

ax.text(

0, -1, # Координаты размещения текста на графике

"Leaky ReLU", # Сам текст

fontsize=12, color="blue", ha="center"

)

a = b

b = a + 24

w1 = w1

w2 = 1.0

if a < frame < b:

state\_dict["fc1.weight"][0][0] = w1 + (w2 - w1) \* (frame - a) / (b - a)

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

a = b

b = a + 40

w1 = w2

w2 = -0.25

if a < frame < b:

state\_dict["fc1.weight"][0][0] = w1 + (w2 - w1) \* (frame - a) / (b - a)

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

a = b

b = a + 10

if a <= frame < b:

pass

a = b

b = a + 40

w1 = w2

w2 = 0.25

if a < frame < b:

state\_dict["fc1.weight"][0][0] = w1 + (w2 - w1) \* (frame - a) / (b - a)

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

a = b

b = a + 12.5

if a <= frame < b:

pass

a = b

b = a + 12.5

b1\_base = b1

b1 = b1

b2 = -50

if a <= frame < b:

bias = b1 + (b2-b1)\*(frame - a)/ (b-a)

state\_dict["fc1.bias"][0] = bias

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

a = b

b = a + 20

b1 = b2

b2 = 25

if a <= frame < b:

bias = b1 + (b2-b1)\*(frame - a)/ (b-a)

state\_dict["fc1.bias"][0] = bias

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

a = b

b = a + 15 #итого = 190

b1 = b2

b2 = b1\_base

if a <= frame < b:

bias = b1 + (b2-b1)\*(frame - a)/ (b-a)

state\_dict["fc1.bias"][0] = bias

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

y = model(x).reshape(-1)

sc.set\_offsets(

torch.column\_stack((x.detach(), y.detach())).numpy()

)

return sc,

# Инициализация графика

def init():

sc.set\_offsets(torch.empty((0, 2)))

return sc,

# Создаём анимацию

from functools import partial

w1 = 0.25

b1 = 0

state\_dict["fc1.weight"][0][0] = w1

state\_dict["fc1.bias"][0] = b1

global text\_w\_b

text\_w\_b = ax.text(

-6, -1.5, # Координаты размещения текста на графике

f"fw = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}", # Сам текст

fontsize=12, color="blue", ha="center"

)

# Обёртка для передачи дополнительных параметров

update\_func = partial(update, w1=w1, b1=b1, text\_w\_b = text\_w\_b)

frames = 190

ani = FuncAnimation(fig, update\_func, frames=frames, init\_func=init, blit=True)

# Сохраняем анимацию

ani.save("weights\_bias\_change.gif", writer="pillow", fps=10)

plt.show()

# 2д визуализация обучения FCNN нейросети под данные

model = FCNN()

state\_dict = model.state\_dict()

state\_dict["fc1.weight"][0][0] = -0.56

state\_dict["fc1.bias"][0] = -0.6

model.load\_state\_dict(state\_dict)

print(f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}")

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from functools import partial

# Инициализация графика

fig, ax = plt.subplots(figsize=(6, 6))

ax.set\_xlim(-100, 100)

ax.set\_ylim(-2, 2)

ax.set\_xlabel("x")

ax.set\_ylabel("y")

# Пустой scatter plot и текст для отображения весов и смещений

sc = ax.scatter([], [], label="Model Predictions")

text\_w\_b = ax.text(

-20, -1.5, # Координаты текста

"", fontsize=12, color="blue", ha="center"

)

# Истинные точки данных

ax.scatter(x\_true.numpy(), y\_true.numpy(), color="red", label="True Data")

ax.legend()

# Обновление графика

def update(frame, tr\_loader, ts\_loader, criterion, opt, model, state\_dict):

# Обучение модели

forward(tr\_loader, criterion, opt)

# Обновление state\_dict для отображения весов и смещения

model.load\_state\_dict(state\_dict)

text\_w\_b.set\_text(

f"w = {state\_dict['fc1.weight'][0][0]:.2f}, b = {state\_dict['fc1.bias'][0]:.2f}"

)

# Предсказания модели

preds, x\_preds = predict(model, ts\_loader)

sc.set\_offsets(torch.column\_stack((torch.tensor(x\_preds, dtype = torch.float32), torch.tensor(preds, dtype = torch.float32))).numpy())

return sc, text\_w\_b

# Инициализация пустого графика

def init():

sc.set\_offsets(torch.empty((0, 2)))

text\_w\_b.set\_text("")

return sc, text\_w\_b

# Параметры анимации

frames = 300

state\_dict = model.state\_dict()

# Частичное применение функции обновления с фиксированными аргументами

update\_func = partial(update, tr\_loader=tr\_loader, ts\_loader=ts\_loader,

criterion=criterion, opt=opt, model=model,

state\_dict=model.state\_dict())

ani = FuncAnimation(fig, update\_func, frames=frames, init\_func=init, blit=True)

# Сохранение и показ анимации

ani.save("training\_animation.gif", writer="pillow", fps=30)

plt.show()

# 3д график выхода сложной FCNN нейросети

def meta\_set\_2d\_pca(n\_samples, x\_pca, pca):

x\_pca\_min = x\_pca.min(axis=0)

x\_pca\_max = x\_pca.max(axis=0)

col\_0 = np.linspace(x\_pca\_min[0], x\_pca\_max[0], n\_samples)

col\_1 = np.linspace(x\_pca\_min[1], x\_pca\_max[1], n\_samples)

grid\_0, grid\_1 = np.meshgrid(col\_0, col\_1)

space = np.stack([grid\_0.flatten(), grid\_1.flatten()], axis=1)

space\_orig = pca.inverse\_transform(space)

return torch.tensor(space\_orig, dtype = torch.float32)

def meta\_data(pca, loader):

x\_pca = pca.fit\_transform(loader.dataset.x)

#print(pca.explained\_variance\_ratio\_)

space\_orig = meta\_set\_2d\_pca(100, x\_pca, pca)

#space\_orig = meta\_set\_2d\_pca(val\_set, 100, val\_set.x.detach().numpy(), pca)

#print(space\_orig.shape[0])

y\_space\_orig = torch.randn(space\_orig.shape[0],1)

val\_meta\_set = dataset(space\_orig, y\_space\_orig)

val\_meta\_loader = torch.utils.data.DataLoader(val\_meta\_set, batch\_size=batchsize, shuffle=False, num\_workers=2)

x\_meta\_pca = pca.fit\_transform(val\_meta\_set.x.detach().numpy())

return x\_pca, x\_meta\_pca, val\_meta\_loader

#@title Визуализация состояния сети

import numpy as np

import torch

import matplotlib.pyplot as plt

import plotly.graph\_objects as go

from mpl\_toolkits.mplot3d import Axes3D

import os

def visual(x\_pca, val\_loader, x\_meta\_pca, val\_meta\_loader, orig=0, inter=0, output\_dir = '/content/graphs', epoch = 1):

if not orig == 0:

predictions = torch.tensor([])

model.eval()

for batch in val\_loader:

with torch.no\_grad():

preds = model(batch[0])

predictions = torch.cat((predictions, preds.detach().flatten()), 0)

predictions = predictions.numpy()

if inter == 1:

fig = go.Figure(data=[go.Scatter3d(

x=x\_pca[:, 0],

y=x\_pca[:, 1],

z=predictions,

mode='markers',

marker=dict(

size=5,

color=predictions,

colorscale='Plasma',

colorbar=dict(title='Network Output')

)

)])

fig.update\_layout(

title='Network Output in PCA Space (Interactive 3D)',

scene=dict(

xaxis\_title='PCA Component 1',

yaxis\_title='PCA Component 2',

zaxis\_title='Predictions'

)

)

fig.show()

if inter == 0:

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

sc = ax.scatter(x\_pca[:, 0], x\_pca[:, 1], predictions, s=20)

cbar = plt.colorbar(sc, ax=ax, pad=0.1, label='Network Output')

ax.set\_title('Network Output in PCA Space (3D)')

ax.set\_xlabel('PCA Component 1')

ax.set\_ylabel('PCA Component 2')

ax.set\_zlabel('Настоящие данные')

plt.show()

predictions\_meta = torch.tensor([])

model.eval()

for batch in val\_meta\_loader:

with torch.no\_grad():

preds = model(batch[0])

predictions\_meta = torch.cat((predictions\_meta, preds.detach().flatten()), 0)

predictions\_meta = predictions\_meta.numpy()

if inter == 1:

fig = go.Figure(data=[go.Scatter3d(

x=x\_meta\_pca[:, 0],

y=x\_meta\_pca[:, 1],

z=predictions\_meta,

mode='markers',

marker=dict(

size=5,

color=predictions\_meta, # Цвет точек соответствует предсказаниям

colorscale='Plasma', # Используем цветовую карту Plasma

colorbar=dict(title='Network Output') # Добавляем цветовую шкалу

)

)])

fig.update\_layout(

title='Network Output in PCA Space (Interactive 3D)',

scene=dict(

xaxis\_title='PCA Component 1',

yaxis\_title='PCA Component 2',

zaxis\_title='Predictions'

)

)

fig.show()

if inter == 0:

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

sc = ax.scatter(x\_meta\_pca[:, 0], x\_meta\_pca[:, 1], predictions\_meta, s=20)

cbar = plt.colorbar(sc, ax=ax, pad=0.1, label='Network Output')

ax.set\_title('Network Output in PCA Space (3D)')

ax.set\_xlabel('PCA Component 1')

ax.set\_ylabel('PCA Component 2')

ax.set\_zlabel('Предсказание')

# Сохранение графика

os.makedirs(output\_dir, exist\_ok=True)

output\_path = f"{output\_dir}/epoch\_{epoch+1:03d}.png"

plt.savefig(output\_path)

plt.show()

plt.close()

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

x\_pca, x\_meta\_pca, val\_meta\_loader = meta\_data(pca, val\_loader)

visual(x\_pca, val\_loader, x\_meta\_pca, val\_meta\_loader, orig=0, inter=0, output\_dir = '/content/graphs', epoch = 1)

многомерные данные

# UMAP

[UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction — umap 0.5 documentation](https://umap-learn.readthedocs.io/en/latest/index.html)

## Гиперпараметры

### Основные гиперпараметры

1. \*\*`n\_neighbors`\*\*

- \*\*Описание\*\*: Количество ближайших соседей, используемых для обучения модели локальной структуры данных.

- \*\*По умолчанию\*\*: 15

- \*\*Рекомендации\*\*:

- Меньшее значение (например, 5) захватывает более локальную структуру.

- Большее значение (например, 50) делает проекцию более глобальной, что помогает выявлять крупные структуры.

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(n\_neighbors=10)

```

2. \*\*`n\_components`\*\*

- \*\*Описание\*\*: Размерность результирующего пространства (обычно 2D или 3D для визуализации).

- \*\*По умолчанию\*\*: 2

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(n\_components=3)

```

3. \*\*`metric`\*\*

- \*\*Описание\*\*: Метрика расстояния для определения близости между точками.

- \*\*По умолчанию\*\*: `'euclidean'`

- \*\*Другие метрики\*\*:

- `'manhattan'`, `'cosine'`, `'correlation'`, `'hamming'`, и другие.

- Можно указать пользовательскую метрику (функцию).

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(metric='cosine')

```

4. \*\*`min\_dist`\*\*

- \*\*Описание\*\*: Минимальное расстояние между точками в низкоразмерном пространстве. Контролирует плотность кластеров.

- \*\*По умолчанию\*\*: 0.1

- \*\*Рекомендации\*\*:

- Меньшее значение (например, 0.01) сохраняет больше локальной структуры, делая кластеры плотнее.

- Большее значение (например, 0.5) раздвигает кластеры.

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(min\_dist=0.2)

```

5. \*\*`spread`\*\*

- \*\*Описание\*\*: Контролирует расстояние между кластерами.

- \*\*По умолчанию\*\*: 1.0

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(spread=1.5)

```

---

### Гиперпараметры обучения

6. \*\*`random\_state`\*\*

- \*\*Описание\*\*: Задает начальное состояние генератора случайных чисел для воспроизводимости результатов.

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(random\_state=42)

```

7. \*\*`learning\_rate`\*\*

- \*\*Описание\*\*: Скорость обучения для оптимизации низкоразмерного представления.

- \*\*По умолчанию\*\*: 1.0

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(learning\_rate=0.5)

```

8. \*\*`n\_epochs`\*\*

- \*\*Описание\*\*: Количество эпох для оптимизации. Если данные очень большие, UMAP может автоматически уменьшить количество эпох.

- \*\*По умолчанию\*\*: 500

- \*\*Пример\*\*:

```python

reducer = umap.UMAP(n\_epochs=300)

```

---

### Пример использования с кастомизацией

```python

import umap

reducer = umap.UMAP(

n\_neighbors=10,

n\_components=3,

min\_dist=0.2,

metric='cosine',

random\_state=42,

learning\_rate=0.5

)

embedding = reducer.fit\_transform(x\_map)

```

Настраивая эти гиперпараметры, ты можешь подогнать UMAP под конкретные данные. Если нужно что-то конкретное, могу помочь подобрать параметры для твоей задачи!

# PCA, UMAP визуализация

## 2 компоненты

import umap

import matplotlib.pyplot as plt

import plotly.graph\_objects as go

x\_map = tr\_loader.dataset.x.detach().numpy()

y\_map = tr\_loader.dataset.y.flatten().detach().numpy()

reducer = umap.UMAP(n\_components=2, random\_state=42)

embedding = reducer.fit\_transform(x\_map)

#метрика похожести

from sklearn.metrics import pairwise\_distances

original\_distances = pairwise\_distances(x\_map)

projected\_distances = pairwise\_distances(embedding)

correlation = np.corrcoef(original\_distances.flatten(), projected\_distances.flatten())[0, 1]

print(f"Корреляция расстояний: {correlation:.4f}")

# Визуализация

fig0 = plt.figure(figsize = (6,5))

plt.scatter(embedding[:, 0], embedding[:, 1], c=y\_map, cmap='Spectral', s=5)

plt.colorbar()

plt.title('UMAP Embedding')

plt.show()

fig = go.Figure(data=[go.Scatter3d(

x=embedding[:, 0],

y=embedding[:, 1],

z=y\_map,

mode='markers',

marker=dict(

size=5,

color=y\_map,

colorscale='Plasma',

colorbar=dict(title='Network Output')

)

)])

fig.update\_layout(

title='3d map of data',

scene=dict(

xaxis\_title='embedding 1',

yaxis\_title='embedding 2',

zaxis\_title='embedding 3'

)

)

fig.show()

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

x\_pca = pca.fit\_transform(x\_map)

print(pca.explained\_variance\_ratio\_)

fig = go.Figure(data=[go.Scatter3d(

x=x\_pca[:, 0],

y=x\_pca[:, 1],

z=y\_map,

mode='markers',

marker=dict(

size=5,

color=y\_map,

colorscale='Plasma',

colorbar=dict(title='Network Output')

)

)])

fig.update\_layout(

title='3d map of data',

scene=dict(

xaxis\_title='embedding 1',

yaxis\_title='embedding 2',

zaxis\_title='embedding 3'

)

)

fig.show()

## 3 компоненты

import umap

import matplotlib.pyplot as plt

import plotly.graph\_objects as go

x\_map = tr\_loader.dataset.x.detach().numpy()

y\_map = tr\_loader.dataset.y.flatten().detach().numpy()

reducer = umap.UMAP(n\_components=3, random\_state=42)

embedding = reducer.fit\_transform(x\_map)

#метрика похожести

from sklearn.metrics import pairwise\_distances

original\_distances = pairwise\_distances(x\_map)

projected\_distances = pairwise\_distances(embedding)

correlation = np.corrcoef(original\_distances.flatten(), projected\_distances.flatten())[0, 1]

print(f"Корреляция расстояний: {correlation:.4f}")

fig0 = plt.figure(figsize = (6,5))

plt.scatter(embedding[:, 0], embedding[:, 1], c=y\_map, cmap='Spectral', s=5)

plt.colorbar()

plt.title('UMAP Embedding')

plt.show()

fig2 = go.Figure(data=[go.Scatter3d(

x=embedding[:, 0],

y=embedding[:, 1],

z=embedding[:, 2],

mode='markers',

marker=dict(

size=5,

color=y\_map,

colorscale='Plasma',

colorbar=dict(title='Network Output')

)

)])

fig2.update\_layout(

title='3d map of data',

scene=dict(

xaxis\_title='embedding 1',

yaxis\_title='embedding 2',

zaxis\_title='embedding 3'

)

)

fig2.show()

from sklearn.decomposition import PCA

pca = PCA(n\_components=3)

x\_pca = pca.fit\_transform(x\_map)

print(pca.explained\_variance\_ratio\_)

fig3 = go.Figure(data=[go.Scatter3d(

x=x\_pca[:, 0],

y=x\_pca[:, 1],

z=x\_pca[:, 2],

mode='markers',

marker=dict(

size=5,

color=y\_map,

colorscale='Plasma',

colorbar=dict(title='Network Output')

)

)])

fig3.update\_layout(

title='3d map of data',

scene=dict(

xaxis\_title='embedding 1',

yaxis\_title='embedding 2',

zaxis\_title='embedding 3'

)

)

fig3.show()